Express Mail No. EL844345660US

PATENT APPLICATION OF

Joseph E. Pentheroudakis, David G. Bradlee
and Sonja S. Knoll

ENTITLED

TOKENIZER FOR A NATURAL LANGUAGE PROCESSING SYSTEM

Docket No. M61.12-0317

TOKENIZER FOR A NATURAL LANGUAGE PROCESSING SYSTEM

The present application is based on and claims the benefit of U.S. provisional patent application Serial No. 60/219,579, filed July 20, 2000, the content of which is hereby incorporated by reference in its entirety.

BACKGROUND OF THE INVENTION

The present invention deals with natural language processing. More specifically, the present invention relates to a tokenizer or tokenizing an input string to accommodate further natural language processing of the input string.

Natural language processing systems are used to develop machine understanding of human expressions. For example, some computer systems attempt to take action based on natural language inputs to the computer system. In order to do this, the computer system must develop an "understanding" of the natural language expression input to the computer. The natural language expressions can be input, for example, as a query in an information retrieval system. The natural language processing system attempts to understand the meaning of the query input by the user, in order to improve precision and recall in the information retrieval task. Of course, there are many other applications for natural language processing, including command

and control, document clustering, machine translation, etc.

One of the principal challenges of a natural language processing system is to identify the boundaries of words in written text. Once the words are identified, the grammar of the particular language being used arranges the words in linguistically significant chunks or constituents. The grammar identifies the hierarchy connecting those constituents and thus constructs a representation for the input sentence.

Many words in written text can be thought of as character strings surrounded by spaces. For example, using a heuristic it can be seen that the preceding sentence contains 15 words and that the last word in the sentence is "spaces". We are able to recognize "spaces" even though that word was written with a period attached (i.e., "spaces."). However, if the sentence were to end in an abbreviation (say, the string "TENN.", for Tennessee), the period would form an integral part of the word. Thus, recognizing when to treat punctuation characters as part of a word and when not to do so is a major challenge for a natural language processing system.

Now consider the string "15MB", which is normally interpreted as "15 megabytes". There are no spaces between the "15" and "MB" and yet the string is normally analyzed as being composed of two

separate words, pronounced "15" and "megabytes", respectively. Also, consider the fact that the grammar will want to know that this construction is the same as the construction for a version written with a space, namely, "15 MB", so that it can treat both versions (with or without a space) identically. Thus, recognizing when to separate digits from alphabetical characters poses another significant challenge for word recognition components of natural language processing systems.

Further consider strings consisting of multiple punctuation characters. Emoticons belong in this class, but so do less glamorous items such as the less than or equal to sign <=, the greater than or equal to sign >=, and the arrow sign: ==> to name a few. It is likely that a natural language processing system will want to treat these as single items. However, distinguishing these from other sequences of multiple punctuation characters, such as, for example, the sequence !)" in the sentence "This is a test (really!)", is also a task that needs to be addressed.

Many other similar difficulties must be addressed as well. For instance, the above examples do not even mention expressions which are highly difficult to interpret, such as "12:00a.m.-4:00p.m.". Further, the above examples do not address other difficult issues, such as electronic mail addresses,

drive path names, and uniform resource locators (URLs).

For the purpose of the present application, the term "token" will refer to any input text flanked by white spaces or by the beginning and/or end of the input string. The term "word" is used to identify the linguistic unit (or units) into which a given token is broken or segmented after undergoing the tokenization process.

Prior tokenizers have suffered from two major problems. First, the tokenization process was performed independently of any knowledge contained in the system's lexical layer. Also, all the knowledge that the tokenizer needed for tokenizing an input string was hard coded in system code. Therefore, prior systems simply implemented the rules used to break input strings apart into tokens without caring whether the segmentation or tokenization made any sense, given the lexical or linguistic knowledge in the lexical layer.

For example, prior systems typically had a rule hard coded which required colons to be separated from surrounding text. Therefore, the example mentioned above "12:00am-4:00pm" would be separated into the following 5 tokens:

12 : 0am-4 : 00pm

Of course, when this expression is handed to the later natural language processing components, it is basically undecipherable.

Prior systems also had such rules as deleting the period at the end of an input string. Therefore, an input string which ended with the terms ". . . at 9:00 A.M." would have its final period deleted, resulting in the token "A.M". In order to recognize this token as a valid lexical word, the lexicon or system dictionary against which the tokens are validated was required to include "A.M" as an entry.

Similarly, such prior systems were not language-independent, by any means. For example, the fact that the English contraction 'll, (as in the word they'll) constitutes a separate word was handled by scanning for a single quote and then scanning for ll following that quote. In effect, this was the logical equivalent of regular expression matching. This meant that separate code had to be added to the system for French, for example, to handle the myriad of contractions present in that language which also use a single quote. The code written had to reflect the content of the lexicon, and had to anticipate what forms were lexicalized. For instance, lexicalized elided form such as m' and l' had to be listed in system code, so that they would be separated from the following words, allowing forms such as aujourd'hui, where the quote is part of the word, to stay together.

Prior tokenizers exhibited still other problems. For example, prior tokenizers required

that hyphenated strings be kept together as a single word. Such a heuristic was adequate for many cases, such as "baby-sitter" or "blue-green", which are most probably single words. However, for a substantial number of cases this approach is inadequate. The hyphen is often used instead of a dash. Therefore, in sentences such as "This is a test-and a tough one at that." the token "test-a" should be presented to the grammar as two separate words, rather than a single word. Because prior implementations of tokenizers did not have access to the lexicon of the language or to any other linguistic knowledge, resolving this at tokenization time was virtually impossible, leading to inappropriate syntactic analysis later in the natural language processing system.

SUMMARY OF THE INVENTION

The present invention is a segmenter used in a natural language processing system. The segmenter segments a textual input string into tokens for further natural language processing.

In accordance with one feature of the invention, the segmenter includes a tokenizer engine that proposes segmentations and submits them to a linguistic knowledge component for validation. The tokenizer engine interleaves proposing segmentations with attempted validation until further segmentation is no longer possible or desired, or until the input

string has been validly segmented. The proposal of segmentations is illustratively done according to a predetermined hierarchy of segmentation criteria.

In accordance with another feature of the invention, the segmentation system includes language-specific data (that is, data defined for a particular language) that contains a precedence hierarchy for punctuation. If proposed tokens in the input string contain punctuation, they can illustratively be broken into subtokens based on the precedence hierarchy. In one illustrative embodiment, the precedence hierarchy is based on the binding properties of the punctuation marks contained therein.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a computer system in which the present invention may be utilized.

FIG. 2 is a more detailed block diagram of a tokenizing system in accordance with one embodiment of the present invention.

FIG. 3 is a flow diagram showing one illustrative embodiment of the operation of the system shown in FIG. 2.

FIG. 4 illustrates a multi-pass embodiment for proposing a segmentation in accordance with one embodiment of the present invention.

FIG. 5 is a flow diagram showing another illustrative embodiment of the operation of the system shown in FIG. 2.

DETAILED DESCRIPTION OF THE ILLUSTRATIVE EMBODIMENTS

FIG. 1 illustrates an example of a suitable computing system environment 100 on which the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

The invention is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

With reference to FIG. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus,

Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer 100. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier WAV or other transport mechanism and includes any information delivery media. The term "modulated data signal"

means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, FR, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way o example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

The computer 110 may also include other removable/non-removable volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 141 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes

to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given

different numbers here to illustrate that, at a minimum, they are different copies.

A user may enter commands and information into the computer 110 through input devices such as a keyboard 162, a microphone 163, and a pointing device 161, such as a mouse, trackball or touch pad. Other input devices (not shown) may include a joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 190.

The computer 110 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a hand-held device, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110. The logical connections depicted in FIG. 1 include a

local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on remote computer 180. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

FIG. 2 is a block diagram illustrating one embodiment of a tokenization system 200 in accordance with the present invention. Tokenization system 200 includes tokenizer engine 202, language-specific data 204 and a linguistic knowledge component 206.

Linguistic knowledge component 206, in turn, includes lexicon lookup engine 208, morphological analyzer 210 and lexicon 212. The morphological analyzer 210 illustratively includes a word morphological component 214, a number morphological component 216 and a punctuation morphological component 218.

Tokenizer engine 202 receives an input string which illustratively includes a series of characters (alpha characters, numeric characters and punctuation marks or emoticons) and attempts to tokenize the input sting into a set of tokens and words. In other words, the present invention attempts to find words (or other linguistically significant units that function as words but are not listed in the lexicon) in the input string, including mixed-character tokens. In doing this, tokenizer engine 202 accesses language-specific data 204. In one illustrative embodiment, language-specific data 204 describes a precedence hierarchy of punctuation characters and also identifies idiosyncratic groupings of complex punctuation tokens. Based upon this information, tokenizer engine 202 proposes a segmentation of the input string (or a portion of the input string) and access linguistic knowledge component 206 to validate the proposed segmentation.

In order to perform validation, each segmentation is submitted to lexical knowledge component 206. Lexicon lookup engine 208 first accesses lexicon 212, which may illustratively be a

computer readable dictionary, or simply a word list, to determine whether the tokens in the proposed segmentation are recognized by, or contained in, lexicon 212. In addition, linguistic knowledge component 206 may include morphological analyzer 210. For example, if lexicon 212 contains only uninflected word forms (i.e., lemmas), then a morphological analysis is desirable to reduce, say the token "brothers-in-law" to the dictionary form "brother-in-law."

Morphological analyzer 210 can also do more than simply convert words to uninflected forms. For example, morphological analyzer 210 also illustratively includes a number morphological component 216 and a punctuation morphological component 218. These two components illustratively convert numbers and punctuation characters to values which will be recognized by lexicon 212 as well.

If a sub-token is successfully looked up in lexicon 212, and thus validated by linguistic knowledge component 206, that sub-token will not be further broken down. Instead, it is simply passed back to tokenizer engine 202 along with an indication that it has been validated. Tokenizer engine 202 then outputs that sub-token for possible further natural language processing.

Linguistic knowledge component 206 also illustratively invokes morphological analyzer 210 to assist in recognizing "virtual" words in the

language (tokens that need to be treated as single words by the system, even though they are not listed in the dictionary). For instance, tokens such as numbers, electronic mail addresses, drive path names, URLs, emoticons, and the like, can be represented as a single word. Morphological analyzer 210 can assist in recognizing each segment as an actual word, or as a virtual word, by identifying it as a virtual word or reducing it to a normalized form for recognition in lexicon 212.

If, after being submitted to linguistic knowledge component 206, the token or sub-token is not validated, it is returned to tokenizer engine 202 for further segmentation. This process continues until no further segmentation is possible at tokenizer engine 202 or until all subtokens have been validated as words, as defined above. If the subtokens are not validated and no further segmentation is possible, tokenizer engine 202 can simply output the subtokens in their finest segmentation form, or can output an error message or other suitable message either instead of, or along with, the subtokens, indicating that validation was not successful.

FIG. 3 is a flow diagram illustrating the operation of system 200 in accordance with one embodiment of the present invention. Tokenizer engine 202 first receives the input string as discussed above. This is indicated by block 220 in

FIG. 3. Tokenizer engine 202 accesses language-specific data 204 to propose a segmentation as indicated by block 222. As briefly discussed above, language-specific data 204 identifies a precedence hierarchy of punctuation characters and idiosyncratic groupings of punctuation tokens. More specifically, the order of segmentations that tokenizer engine 202 attempts is illustratively controlled by the punctuation hierarchy in language-specific data 204. This hierarchy lists the relative binding properties of punctuation characters in the language. For example, the complex English token "brother-in-law/sister-in-law" contains a slash and several hyphens. If tokenizer engine 202 were to first attempt a segmentation based on the hyphens, the following nine token list would result:

Brother - in - law/sister - in - law

This segmentation would thus miss looking up the strings "brother-in-law" and "sister-in-law" in lexicon 212. Further, the token law/sister would be unrecognizable. However, if language-specific data 204 postulates that the hyphen binds more tightly than the slash, then tokenizer engine 202 will first try breaking at the slash, rather than the hyphen. This yields the following three tokens:

Brother-in-law / sister-in-law

Thus, language-specific data 204 may simply be a list of punctuation characters, or other characters, listed in order of their binding

properties. Tokenizer 202 then simply determines whether any of those punctuation or other characters reside in the token, and attempts to break or segment the tokens at the punctuation or other characters which bind most loosely first. Tokenizer 202 then simply progresses through the list attempting to break the token at more tightly binding characters at each step until validation is achieved.

Once a segmentation has been proposed by tokenizer engine 202, based on language-specific data 204, it is provided to linguistic knowledge component 206. Lexicon lookup engine 208 then access lexicon 212 to determine whether the token or sub-token provided by tokenizer engine 202 exists in the lexicon. This is indicated by block 224 in FIG. 3. If the token or sub-token is found in lexicon 212, it is validated and this is indicated to tokenizer engine 202 by linguistic knowledge component 206. This is indicated by blocks 226 and 228 in FIG. 3.

It should also be noted that, in the embodiment in which lexicon 212 is a dictionary, additional information regarding the word which has been located can also be provided to tokenizer engine 202. In that case, the additional information provided along with the validated token or sub-token (i.e., the word) is useful for additional natural language processing steps which may be undertaken in downstream analysis.

If, at block 226, lexicon lookup engine 208 is unable to locate the token or sub-token in lexicon 212, lexicon lookup engine 208 can optionally access morphological analyzer 210 to further analyze the token or sub-token, morphologically, as described above. This is indicated by block 230 in FIG. 3. If, after morphological analysis, the token or sub-token is located in lexicon 212, it is again validated and this is indicated to tokenizer engine 202. This is illustrated by blocks 232 and 228 in FIG. 3.

However, if, even after a morphological analysis, the token or sub-token is still unfound in lexicon 212 (or a virtual lexicon containing virtual words), this is also indicated to tokenizer engine 202 which then conducts additional segmentation steps and proposes a new segmentation to linguistic knowledge component 206. This is indicated by blocks 232 and 222 in FIG. 3. This process continues recursively until each token and sub-token has been validated by linguistic knowledge component 206, or until tokenizer engine 202 is unable to conduct further segmentations.

FIG. 4 illustrates one embodiment in which system 200 proposes various segmentations as indicated by block 222 in FIG. 3. FIG. 4 illustrates that, in one illustrative embodiment, system 200 analyzes an entire input string (typically a sentence) by making multiple passes through the input string and validating portions of the input string with each pass, if

possible, until the entire string has been validated, or until no further segmentations can be proposed.

More specifically, in the embodiments illustrated in FIG. 4, tokenizer engine 202 illustratively makes a first pass through the input string, simply separating the input string into tokens delineated by white spaces. Tokenizer engine 202 then provides each of the tokens generated from the first pass to linguistic knowledge component 206 to determine whether any of the tokens in the input string can be validated. Thus, as with a large portion of most input strings, system 200 will simply separate the input string into the various words in the input string and those words will be recognized in lexicon 212 and immediately validated. This is indicated by blocks 234 and 236 in FIG. 4.

In the second pass, tokenizer engine 202 accesses a list of emoticons and multiple punctuation characters contained in language-specific data 204, which are to be maintained together. Such characters may include, for example, ==>, <==, =>, ☺ etc. Tokenizer engine 202 then passes these subtokens to linguistic knowledge component 206 which attempts to validate them by locating them, as virtual words, in lexicon 212. This is indicated by blocks 238 and 240 in FIG. 4.

It should be noted that this pass is illustratively conducted prior to applying the precedence hierarchy in pass 3. This is because

applying the precedence hierarchy of punctuation begins to explode the tokens into subtokens based on punctuation characters. If it is desired that any of these punctuation characters be maintained as a single unit, that pass should be conducted before applying the precedence hierarchy. Therefore, attempting to recognize emoticons and multiple punctuation characters is conducted in pass 2, so that such characters can be validated, if desired, prior to breaking the token into subtokens based on punctuation.

In pass 3, tokenizer engine 202 recursively applies the precedence hierarchy of punctuation from language-specific data 204. In other words, the token under analysis by tokenizer engine 202 may contain one or more punctuation marks. The token is broken into subtokens at the punctuation marks according to the precedence hierarchy of punctuation contained in language-specific data 204. Each of the subtokens is then recursively submitted by tokenizer engine 202 to linguistic knowledge component 206 for validation. This is indicated by blocks 242 and 244 in FIG. 4. For exemplary purposes, one illustrative embodiment of a specification of expected tokenization results is attached hereto as appendix A. The appendix specifies how a variety of punctuation characters are to be treated and thus specifies the hierarchy.

In pass 4, tokenizer engine 202 segments each token by splitting numeric and alpha characters

into different parts and then attempting to validate the numeric characters, by themselves, and the alpha characters, also by themselves. For example, the token may be 15NOV98. Of course, in the English language this is understood to be the date November 15, 1998. Therefore, in pass 4, tokenizer engine 202 segments this token into the following three subtokens:

15 Nov 98

Each of these subtokens is then submitted to linguistic knowledge component 206 which illustratively analyzes them by doing a lookup in lexicon 212, and by optionally invoking the word and number morphological analyzer components of morphological analyzer 210. In this example, each of the three subtokens will be recognized as a valid sub-token, and this will be indicated to tokenizer engine 202. Engine 202 can then pass these subtokens downstream to further natural language processing steps which "understand" that the segmentation of these subtokens, in this way, identifies the date November 15, 1998.

Pass 4 will also correctly segment numerous other combinations of alpha and numeric characters, such as 15MB, which is commonly understood to mean 15 megabytes. As with the previous example, tokenizer engine 202 will split this into the two tokens 15 and MB, and those tokens will be submitted for validation to linguistic knowledge component 206 which will, in turn, validate those subtokens. Pass 4 is illustrated by blocks 246 and 248 in FIG. 4.

Finally, tokenizer engine 202 will attempt to recursively reassemble unvalidated subtokens which were previously segmented. Therefore, if, after pass 4, there are still subtokens which have not been validated by lexical knowledge component 206, tokenizer engine 202 recursively attempts to reassemble those subtokens into different tokens and submits each attempted segmentation to lexical knowledge component 206 for validation. This is illustratively done by attempting to join invalid subtokens to the subtokens which lie to their right and then to their left. Reassembling subtokens is illustrated by blocks 250 and 252.

FIG. 5 is a flow diagram illustrating another embodiment for proposing and validating segmentations, as conducted by tokenizer engine 202. tokenizer engine 202 first receives the input string as indicated by block 260. Then, beginning at the left of the input string, tokenizer engine 202 scans to the right until it finds a space, or the end of the input string. This is indicated by block 262. Once this is located, the portion of the input string which has been scanned is identified as a token and is submitted to linguistic knowledge component 206 for validation. Of course, as discussed above, morphological analyzer 210 can optionally be invoked by lexicon lookup engine 208, in order to validate the token. This is indicated by block 264 in FIG. 5.

Linguistic knowledge component 206 then performs a lookup in lexicon 212 and possibly more validation steps by invoking morphological analyzer 210 to determine whether the token is valid. This is illustrated by block 266. If the token is valid, this is indicated to tokenizer engine 202 and tokenizer engine 202 begins at the left side of the next token as indicated by block 268. Again, tokenizer engine 202 identifies the next token as the portion of the input string scanned until a space is located.

However, if, at block 266, the token provided to linguistic knowledge component 206 is not validated, then tokenizer engine 202 determines whether the token is all alpha characters or all numeric characters. This is indicated by block 270. If the token is either all numbers, or all alpha characters, then processing of that token is complete, and the token is treated as an unrecognized word. However, if the token is not either all alpha characters, or all numbers, then tokenizer engine 202 determines whether the token includes final punctuation. This is indicated by block 272.

Determining whether the token includes final punctuation can be done in any number of suitable ways. For example, tokenizer engine 202 can simply determine whether the last character in a token is a punctuation mark and whether the next character in the input string is a space, followed by a capital letter. Of course, there are many different ways to determine whether a

token includes a final punctuation mark, and any suitable way can be used.

In any case, if, at block 272 it is determined that the token includes a final punctuation mark, then the final punctuation mark is split off from the remainder of the token as indicated by block 274. The remaining token and the final punctuation are recursively submitted to linguistic knowledge component 206 for validation. For example, with the primary portion of the token (the subtoken not including the final punctuation mark) lexicon lookup engine 208 performs a lookup in lexicon 212 and invokes word morphological component 214 in morphological analyzer 210, as necessary, in order to validate the subtoken. The final punctuation mark will also be provided by engine 208 to punctuation morphological component 218 of morphological analyzer 210 in an attempt to validate the punctuation mark. If these subtokens are validated, as indicated by block 276, processing returns to block 268 where processing of the next token is started.

However, if, at block 276, the subtokens remaining after the final punctuation is split off are not validated, or if at block 272 it is determined that the token does not include final punctuation, then tokenizer 202 determines whether the subtokens include alpha characters and numeric characters. This is indicated by block 278. If so, the alpha characters are split away from the numeric characters, in a

similar fashion as that described with respect to pass 4 in FIG. 4. This is indicated by block 280 in FIG. 5. Each of these subtokens is again recursively submitted to linguistic knowledge component 206 to determine whether the alpha subtokens and the numeric subtokens are valid. Of course, lexicon lookup engine 208 can invoke the word morphological component 214 and the number morphological component 216 in morphological analyzer 210 in order to make this determination.

If the alpha subtokens and the numeric subtokens are validated as indicated by block 282, then the processing again reverts to block 268 for processing of the next token. However, if at block 278 the token does not include alpha and numeric characters, or if at block 282 the alpha and numeric subtokens are not validated, tokenizer engine 202 determines whether the token includes an emoticon. This is indicated by block 284. Tokenizer engine 202 accesses language-specific data 204 to determine whether emoticons exist in the token. If so, the emoticons are separated out from the remainder of the token as indicated by block 286. The emoticons and the remaining subtokens are then submitted for validation at block 206. This is indicated by block 288. If they are validated, processing reverts to block 268.

However, if at block 284, the token does not include emoticons or if at bock 288, the subtokens are not validated, then tokenizer engine 202 determines whether the token includes an edge punctuation mark

(i.e., a punctuation mark either at the first character position of the token or at the last character position of the token). This is indicated by block 290. If the token does include one or more edge tokens, tokenizer engine 202 accesses language-specific data 204 for a precedence hierarchy in splitting off the edge tokens. In other words, tokenizer engine 202 works from both the right and left sides of the token, where punctuation resides, and splits the punctuation marks off from the remainder of the token in a precedence fashion, as indicated by language-specific data 204. This is indicated by block 292. The subtokens which are derived after splitting off the edge punctuation in accordance with each step in the precedence hierarchy are recursively submitted to linguistic knowledge component 206 for validation. Of course, as discussed above, linguistic knowledge component 206 can invoke any or all of the morphological components of morphological analyzer 210 as well as lexicon 212, in order to validate the subtokens.

If the subtokens are validated, as indicated by block 294, then processing again reverts to block 268. However, if at block 290 it is determined that the token does not include edge punctuation, or the subtokens after splitting off the edge punctuation are not valid, as determined at block 294, then tokenizer engine 202 determines whether the token includes any interior punctuation marks. This is indicated by block 296. It can thus be seen that tokenizer engine 202

implements a preference for maintaining internal punctuation marks together, and attempts to split off edge punctuation marks first. This is useful because it is believed that edge punctuation marks do not generally bind as tightly as interior punctuation marks. For example, assume that a sentence ends with the phrase … my relation (brother-in-law). It can be seen that even after the final punctuation is split off, the interior punctuation marks bind more tightly than the edge punctuation marks. Therefore, this would successfully be split into the three tokens (brother-in-law).

Of course, the token may include a plurality of different interior punctuation marks. Therefore, tokenizer engine 202 accesses a precedence hierarchy of punctuation in language-specific data 204 and begins the segmentation of the token into subtokens based on that precedence hierarchy. This is indicated by block 298. Each of the subtokens is then recursively submitted to linguistic knowledge component 206 for validation. As with prior steps, the subtokens are processed and validated, if possible. This is indicated to tokenizer engine 202.

In block 298, tokenizer engine 202 illustratively simply marches down the precedence list of punctuation and segments the token into subtokens based on that precedence list. However, a number of special cases can be handled differently. For example, if tokenizer engine 202 finds an apostrophe (such as

'll or 's) in the token, then tokenizer engine 202 can perform a number of special handling steps. In one illustrative embodiment, the apostrophe is handled by trying to append it to the subtoken on both sides of the apostrophe. Therefore, for the term "they'll" tokenizer engine 202 may first try to segment by attempting to validate the subtoken they'. Tokenizer engine 202 can then attempt to validate the subtoken 'll as corresponding to the word "will". Of course, in that embodiment, linguistic knowledge component 206 will likely validate the subtoken 'll and will not validate the subtoken they'. Similarly, for a token such as O'Toole's tokenizer engine 202 will break the token into subtokens at the apostrophes and attempt to attach the apostrophes to the subtokens coming before them, and after them, until valid subtokens are obtained. Therefore, in one illustrative embodiment, the token O'Toole's will be broken into the following three valid subtokens O' Toole 's.

If, at block 296 it is determined that the token does not include interior tokens, or if at block 300, it is determined that one or more of the subtokens remaining after segmentation are not valid, then tokenizer engine 202 either simply outputs an error message associated with those subtokens indicating that valid segmentation has not been obtained, or tokenizer engine 202 can attempt to recursively reassemble the remaining subtokens and resubmit them for validation. This is indicated by block 302. Reassembling the

subtokens can be done in a similar fashion to pass 5 mentioned in FIG. 4 above.

It should also be noted that tokenizer engine 202 performs the same regardless of the language it is used with. The only things that change with different languages are possibly language-specific data 204 (since certain languages may have different precedence hierarchies), lexicon 212, and possibly portions of morphological analyzer 210. However, the tokenizer engine 202, itself, does not change.

Similarly, tokenizer engine 202 can optionally be modified for different languages quite easily. For example, in the Greek language, alpha characters and Arabic numerals are never used together as part of a single token. Therefore, looking for alpha and numeric characters together need not be undertaken. In that case, the procedure indicated by block 278 in FIG. 5 and pass 4 in FIG. 4 can simply be turned off in tokenizer engine 202, without modifying the code for tokenizer 202 in any significant way.

It should further be noted that, even though portions of the linguistic knowledge component 206 may change with the different languages, the call from tokenizer engine 202 to linguistic knowledge component 206 need not change.

It can thus be seen that the present invention may illustratively include a precedence hierarchy of the binding properties of different characters which are encountered in tokenization.

The present invention can also illustratively interleave application of the precedence hierarchy with accessing the linguistic knowledge components of the natural language processing system in order to validate different segmentations prior to submitting those segmentations for further natural language processing.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

APPENDIX A

This appendix represents the set of guidelines and expected results developed for testing the accuracy and performance of the tokenizer for English.

1 Introduction

Tokenization constitutes the first step in the creation of a parse for a given sentence. It separates the sentence into tokens. Tokens are typically the elements between the space characters.

The tokenizer sends potential tokens to morphology. Morphology then determines whether the potential token is valid. If it is, morphology creates a lexical record for it, and notifies the tokenizer, which does then not break the validated token further up. The tokenizer creates only one set of tokens, each token giving rise to one lexical record.

The results of tokenization in the examples below correspond to the expected result. Section 2 describes the general principles underlying the tokenizer. Section 3 discusses a set of constructions that the tokenizer must handle.

2 Expected results : rules

The pre-conditions and rules given in this section describe the expected results for the tokenizer.

Pre-condition 1
Tokens do not span blank characters, i.e. tokens never contain blank characters.

Exceptions:
- Ellipses that contain spaces constitute tokens.

Pre-condition 2
Tokenization does not separate sequences of alphabetic characters. Similarly, it does not separate sequences of digits

Example:
Consider the sequence:
Fdsj9sfjk)(fjd453
Potential tokens include: Fdsj, 9, sfjk,), (, fjd, 453, and any combination of these (when adjacent) (such as Fdsj9sfjk, Fdsj9), but not j9s, fjd45 or Fdsj9s.

2.1 General Rule

A string or part of a string constitutes a token if and only if
- it is valid (see definitions below),
- it is not part of a larger valid string

In the example below, the tokenizer does not break up "baby-sitter" into "baby", "-", and "sitter" because "baby-sitter" is a valid string: the strings "baby" and "sitter" do not constitute tokens because they are part of the larger valid string "baby-sitter". (In the examples to follow, the output of the tokenizer is bolded:)

1) The baby-sitter is in the garden.
The baby-sitter is in the garden .

A string is considered valid if it is in the lexicon or it can be reduced to a word found in the lexicon.

The tokenizer breaks up 's from the preceding word, but does not break up 's further, since 's is in the dictionary.

2) John's daughter is here.
John 's daughter is here .

The tokenizer does not break up "pre-" because "pre-" is in the lexicon.
The tokenizer does not break up "post-tests" because "post-tests" is derived from "test" (derivational morphology).

3) the pre- and post-tests.
the pre- and post-tests .

Note on affixes:

There are two types of affixes:

- Type 1: affixes that can attach to groups of words: these should be separated by the Tokenizer
Example: one million-plus
"Million-plus" should be separated by the Tokenizer.

- Type 2: affixes that only attach to single words: these should not be separated by the Tokenizer
Example: happy-ish
"Happy-ish" should be recognized by morphology and thus not separated by the Tokenizer.

A string is also considered valid if it consists of a sequence of identical punctuation characters that should stay together; for English, that list contains the characters -` = % < > # @ * + . , & : | ! ?

The tokenizer keeps multiple ? together.

4) Was he kidding ????
Was he kidding ????

Only the "!!" in the "!?!!" string stays together, since it is the only sequence of more than one character that is valid.

5) This is a test!?!!
This is a test ! ? !!

A string is also considered valid if it consists of a sequence of punctuation characters that form a single unit(such as an emoticon, an arrow or a string such "+=" and ""). Strings that belong to this class include -> -->> += +/- ==> => --> ---> ===> " '' %-) :-) :-)) ;-) ;-> :) :-> :-(:-< :> and :<

The tokenizer keeps "''" together.

6) It measured 3''.
It measured 3 '' .

The tokenizer keeps "+=" together.

7) X += 3
X += 3 .

The strings "->" and "-->" are kept together.

Scoring tables have good -> better --> best.

Scoring tables have good -> better --> best .

The string ":-)" is kept together.

8) I like Ike :-)

I like Ike :-) .

The string "-->" is kept together, but separated from the ".

9) "-->"

"-->".

A string is also considered valid if it can be interpreted as being a decimal number and/or a number with thousands separators.

The decimal number "3.5" is not separated.

10) It measured 3.5 inches.
It measured 3.5 inches .

The number "100,000" is not separated.

11) He had a 100,000 teacher perfect sentences.
He had a 100,000 teacher perfect sentences .

A string is also considered valid if it can be interpreted as being a URL, an email address or a pathname

The tokenizer recognizes URLs.

12) They ran Higher Source (www.concentric.net/~Font/), a Web-design firm in Rancho Santa Fe.
They ran Higher Source (www.concentric.net/~Font/) , a Web - design firm in Rancho Santa Fe .

13) For a Gregory Nava filmography, consult the Internet Movie Database (us.imdb.com/~M/person-exact?Nava%2C%20Gregory).
For a Gregory Nava filmography , consult the Internet Movie Database (us.imdb.com/~M/person-exact?Nava%2C%20Gregory) .

The tokenizer recognizes email addresses.

14) His address is Chr@xxx.eee.com.
His address is Chr@xxx.eee.com .

The tokenizer recognizes pathnames.

15) You can find the documentation in c:\root\directory.
You can find the documentation in c:\root\directory .

A string of alphas and punctuation characters not in the punctuation list is also considered valid.

The punctuation list does not contain the "_" character. "_the" and "best_" are valid strings.

16) I went to _the best_ restaurant yesterday.
I went to _the best_ restaurant yesterday .

Similarly, "Mar_key" is a valid string.

17) Hello, my name is _____ with Mar_key research.
Hello , my name is _____ with Mar_key research .

Periods are broken off from strings only at the end of sentences.

18) Call it document.txt or something.
Call it document.txt or something .

19) Read section 2.3.8.
Read section 2.3.8 .

20) "10a.m". to "2p.m".
" 10a.m " . to " 2p.m " .

21) My relatives live in Pleasantville, NY.50 miles from the Mexican border.
My relatives live in Pleasantville , NY.50 miles from the Mexican border .

22) Does Michael A.Johnson live here?
Does Michael A.Johnson live here ?

23) J.K. Smith is happy.
J.K. Smith is happy .

Exceptions:

1.Today is Friday.
1. Today is Friday .

He was here. . . and I was there.

He was here . . . and I was there.

He was here. . and I was there.
He was here . . and I was there.

A single character that belongs to the punctuation list is also considered valid. Punctuation markers that belong to this class include: (){}[]<>" # !?; @£¥$¢€° % / - , . : '`' & +*× ©® = ^ ~

Although the "_" is not in the punctuation list, "," is, so the comma is separated from the sequence of underscores in the example below:

24) Hello, my name is _____, with Mar_key research.
Hello , my name is _____ , with Mar_key research .

A string of alphas (with optional period(s)) is also considered valid.

Although "He", "e" and "elp" are not in the lexicon, the tokenizer separates the string "He-e-elp!".

25) He-e-elp!
He - e - elp !

Similarly, although "R.I" is not in the lexicon, the tokenizer separates "R.I".".

26) My relatives live in Pleasantville, R.I".
My relatives live in Pleasantville , R.I ".

2.2 Alpha-numeric strings

The tokenizer will break up a string containing alphas (with optional period(s)) and nums if and only the original string can also be written broken up (see typical cases below). The resulting alpha string (with optional period(s)) must be valid.

The tokenizer breaks up a num followed by "km".

27) He drove 5km.
He drove 5 km .

The tokenizer separates "4p.m." into "4" and "p.m.".

28) He arrived at 4p.m.
He arrived at 4 p.m. .

The tokenizer separates "1154B.C." into "1154" and "B.C.".

29) He was born in 1154B.C.
He was born in 1154 B.C. .

The tokenizer separates "x789" into "x" and "789".

30) You can call him at 333-4444 x789.
You can call him at 333 - 4444 x 789.

2.3 Sentence-final Punctuation Rule

Every sentence must end in a sentence-final punctuation, including the following characters: . : ? !. If the original sentence does not end in a sentence-final punctuation, or if it ends in an abbreviation, a period is introduced.

A period is introduced at the end of this sentence, since there was none to begin with.

31) He is very funny
He is very funny .

The tokenizer adds a period to sentences ending in an abbreviation

32) My relatives live in Pleasantville, Penn.
My relatives live in Pleasantville , Penn. .

The tokenizer adds a period to sentences ending in a paren.

33) John lived in the "lower 48."
John lived in the " lower 48 . " .

The tokenizer adds a period to sentences ending in an ellipsis.

(Double period ellipsis)
34) He is sad..
He is sad .. .

(Triple period ellipsis)

35) He is so funny...
He is so funny ... .

(Ellipsis character)

36) He is sad…
He is sad … .

(Others)

37) He is sad….
He is sad ... .

(… and abbreviation)

38) He lives in Penn..
He lives in Penn. .. .

If the last element of a sentence is ambiguous between being an abbreviation with a period or not, there are two possibilities:

- either the form without the period is preferred. The tokenizer does not need to add a sentence-final period, or
- both entries are recognized.

The tokenizer allows each language to develop its own heuristics in deciding which alternative to use.

3 Expected results : special cases

This section covers different types of constructions that must be handled by the tokenizer. They are formulas, numbers, alphanumerics, complex sequences, sentence initial tokens, Web addresses, Email addresses, URLs, multiple punctuation marks.

3.1 Hyphenated compounds

Hyphenated compounds that are not found in the lexicon are broken up by the tokenizer.

"baby-sitter" is in the lexicon.

39) The baby-sitter is in the garden.
The baby-sitter is in the garden.

"blue-green" is not.

40) I have a blue-green sweater
I have a blue - green sweater .

Strings that contain more than one hyphen are separated into word, hyphen, word, hyphen etc. tokens as long as they don't contain hyphenated words (validated by morphology). .

3.2 Formulas

41) 2+2=4
2 + 2 = 4 .

42) 37-5 = 32.
37 - 5 = 32 .

3.3 Numbers

The tokenizer keeps decimal numbers together.

43) It measured 3.5 inches.
It measured 3.5 inches .

The tokenizer keeps numbers with thousands separators together.

44) He had a 100,000 teacher perfect sentences.
He had a 100,000 teacher perfect sentences .

BUT the tokenizer does not keep numbers in scientific notation together.

45) An erg of energy has a mass of 1.1x10^-21 grams.
An erg of energy has a mass of 1.1 x 10 ^ - 21 grams.

BUT the tokenizer does not keep negative numbers together.

46) The temperature is -5 degrees.
The temperature is - 5 degrees .

BUT the tokenizer does not keep fractions together.

47) It costs 2 1/2 to 3 dollars
It costs 2 1 / 2 to 3 dollars .

BUT the tokenizer does not keep ratios together.

48) The student/teacher ratio is 25:1 in Chicago.
The student / teacher ratio is 25 : 1 in Chicago .

3.4 Alphanumerics

Strings that contain both numbers and letters are not broken up, if they don't make sense separately.

An example.

49) The code is 24MBT.
The code is 24MBT .

If the string contains a number followed by a unit (of measure, etc.), it is broken up.

50) It is a 5km run.
It is a 5 km run .

51) It's a 24MB.
It 's a 24 MB .

3.5 Complex sequences

52) His speed is 5km/hr.
His speed is 5 km / hr .

53) Send it to 1421 E. Union St., Apt.#3.
Send it to 1421 E. Union St. , Apt. # 3 .

54) $3-a-share
$ 3 - a - share .

55) 1-(425)-882-8080
1 - (425) - 882 - 8080 .

56) 1(425)882-8080
1 (425) 882 - 8080 .

57) It was from an article in National Geographic (Volume 11: 12-15).
It was from an article in National Geographic (Volume 11 : 12 - 15) .

58) This is 4:40 pm-5:00 pm.
This is 4 : 40 pm - 5 : 00 pm .

59) The vote was 295-136-better than a two-thirds majority-which means that opponents of the ban must once again depend on the Senate to sustain a presidential veto.

The vote was 295 - 136 - better than a two - thirds majority - which means that opponents of the ban must once again depend on the Senate to sustain a presidential veto .

3.6 Web addresses, Email addresses, URLs

Any web address, email address or URL remains intact after tokenization.

60) "You can find more information at http://www.fas.harvard.edu/~art/shepard.html."
" You can find more information at http://www.fas.harvard.edu/~art/shepard.html . " .

61) (Visit the Whitney site at www.echonyc.com/~whitney/.)
(Visit the Whitney site at www.echonyc.com/~whitney/ .) .

62) They ran Higher Source (www.concentric.net/~Font/), a Web-design firm in Rancho Santa Fe.
They ran Higher Source (www.concentric.net/~Font/) , a Web - design firm in Rancho Santa Fe .

63) Please visit my homepage at http://www.aa.net/~allaboutme.
Please visit my homepage at http://www.aa.net/~allaboutme .

64) For a Gregory Nava filmography, consult the Internet Movie Database (us.imdb.com/~M/person-exact?Nava%2C%20Gregory).
For a Gregory Nava filmography , consult the Internet Movie Database (us.imdb.com/~M/person-exact?Nava%2C%20Gregory) .

Note that web addresses, email addresses or URLs that contain a blank character do not constitute a token: that would violate pre-condition 1.

65) The Riverfront Times joins in the Ray conspiracy-theory frenzy (home.stlnet. com/~cdstelzer/mlk3.html), as does ParaScope (www.para scope.com/mx/luther1.htm).
The Riverfront Times joins in the Ray conspiracy - theory frenzy (home.stlnet . com / ~ cdstelzer / mlk3.html) , as does ParaScope (www.para scope.com/mx/luther1.htm) .